# Your Spectrum

Časopis pravého Spectristy

YS #10: srpen '99






**YOUR SPECTRUM 10/99**
časopis určený výhradně pro uživatele počítačů ZX Spectrum a kompatibilních

**Distribuce, předplatné:**
8BitCompany Publishing
Tomáš Modroczki
Pražská 2532
438 01 Žatec
Česká republika
**tel.:** 0602/472579

**Adresa redakce:**
8BitCompany
Martin Blažek
Luční 4570
760 05 Zlín
Česká republika
**tel.:**0603/543256

**Internet:** www.8bc.com
**e-mail:** 8bc@publikum.cz

**Redakční rada:**
Martin Blažek-Blažko/systems **-BLS-**
Jan Kučera-Last Monster **-LMN-**
Tomáš Modroczki-A. I. D. S. **-AIDS-**
Slavomír Lábsky-Busysoft **-BUSY-**
Rudolf Kozel **-STRY-**



**Obsah YS 10/99:**



**Toto číslo je věnováno Karlu Gottovi k jeho šedesátinám. Karle, ať Ti to stále zpívá alespoň do 128 let.**





# Úvodní blekot

**Vážení Spectristé,**

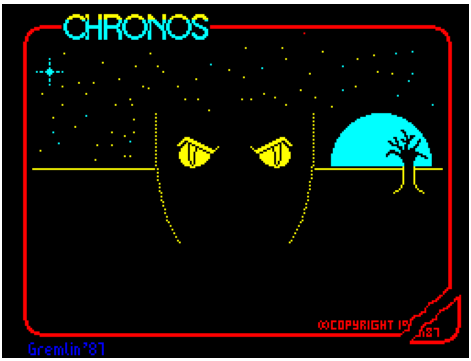
máme tu pro vás další číslo. Ještě jsem ani nedočetl předchozí dvojčíslo a už je tu čas na další vydání.
Máme pro vás pár novinek, které jistě potěší každého Spectristu. Takže,... hurá do toho.
První a největší změnou je rozšíření redakční rady o mě, což by mělo mít za následek pravidelnější vydávání a lepší kvalitu tisku. Aby bylo stále o čem psát, tak bych vás chtěl tímto požádat, abyste nám napsali, co vám v YS chybí a co máme změnit. Taky můžete posílat zajímavé články, návody, schémata a taky komixy. Pokud můžete, tak e-mailem a nebo na disketě poštou. Protože si asi vezmu na starost hry, tak taky posílejte náměty, na jaké hry máme otisknout návody/recenze a pokud něco umíte hrát, tak se o to podělte s námi. Dále se pokusím získat nějaké tajné informace z 8BC-Research Division a taky snad v příštím čísle obrázky z úvodního dema The Devil Inside, proslýchá se, že TDI bude HDD version a to natvrdo s videama.
Taky se zeptáme Busyho co nový SFS.
No a k tomuto číslu...
Jako obvykle recenze na hry, Busyho lekce strojáku, zázraky v basicu, a spoustu a spoustu dalších věcí. No prostě YS jak má být.
Takže si tohle číslo pěkně užijte a nezapomeňte psát, kontakt s čtenáři je pro nás naše jediná odměna.

**-STRY-**

# GAMESY

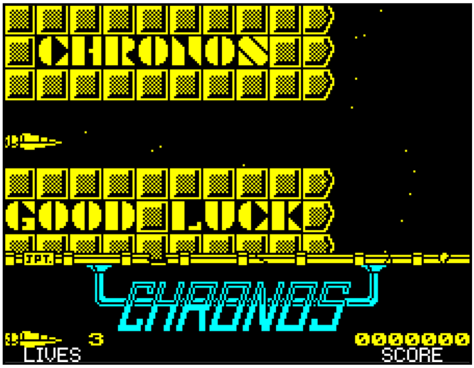
## CHRONOS

**V této recenzi vám nabízíme skvělou hru z roku 1987, kterou vydala firma MASTERTRONIC. Autoři Steven Tatlock - program, John Tatlock - grafika a Tim Follin - hudba vytvořili jednu z klasických střílleček na ZX-Spectrum a nazvali ji: CHRONOS - A TAPESTRY OF TIME.**




*úvodní obrazovka hry*

Tvým úkolem je zachránit CHRONOSe, Pána času. Aby jsi tento úkol mohl splnit, nasedáš do malého vesmírného korábu a vydáváš se na cestu přes šest levelů a jak už to u střílleček bývá, musíš sestřelit vše, co ti přijde do cesty. Hra je velmi pěkně provedena. Grafika se mění každý level, přibývají nepřátelé a zvětšuje se obtížnost. Ovládání tvého vesmírného plavidla je velmi plynulé a dostatečně rychlé, k tomu, aby jsi zvládl všechny nástrahy, které na tebe čekají.




*zde začíná Tvá vesmírná pouť*

Přiznám se, že CHRONOS je moje nejoblíbenější střílečka na ZX-Spectrum vůbec. Výborně se hraje, není příliš těžká, takže při troše cviku ji můžete dorát i bez pouků a navíc obsahuje snad nejlepší hudbu, která kdy byla napsána pro ZX48 (ULA SOUND MACHINE). ULA zde generuje neuvěřitelných pět kánálů. Na ZX128 je vše doplněno ještě o bicí, které hraje AY-ka - prostě paráda. Jeden můj kolega (bývalý atarista), byl tak nadšený touto hrou, že ji projel třicetsedmkrát a stále pokračuje. Tvrdí, že hra je stále obtížnější a obtížnější.

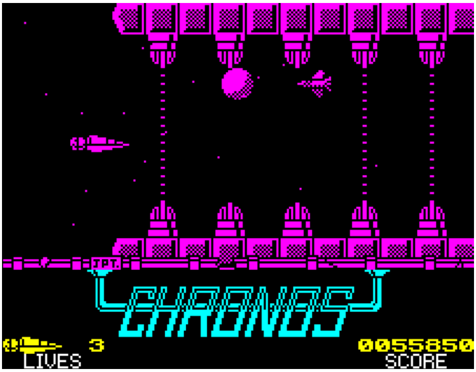



*nejnáročnější část hry-zapovězené jeskyně*

Určitě si zahraj tuto hru, stojí to za to. Pokud se dostaneš až na konec, přivítá tě osobně CHRONOS a promluví k tobě:

"Záchránil jsi mne, jsem ti nekonečně vděčen, ale nekonečno pro mne znamená málo. Jsem

CHRONOS, Pán času. Tvá válka pro mne nebyla nic jiného než zrnko prachu, ačkoliv jsem do ní byl zatažen. Ve své pošetilosti jsi zničil jednu z mnoha bran do zajatecké dimenze, což způsobilo díru v čase, kterou musíš vyplnit. Nyní je ti souzeno bojovat tuto poslední bitvu až do smrti. Sbohem..."




*laserové centrum*

Hra mimojiné obsahuje také mnoho pozdravů a cheatů, zkuste si některé z nich, podepište se do tabulky HIGH-SCORE jako:

nemesis, design design, jing it baby (aktivuje megalaser, který si můžete zapnout v menu!), peter gough, mike follin, tim follin, mark wilson, the thug, fuck, agent x, chronos.




*závěrečný proslov*

A nakonec několik užitečných pouků:
Počáteční počet životů: POKE 53407,životy
Autofire: POKE 26987,201
Nekonečné životy: POKE 56909, 0

**-LMN-**

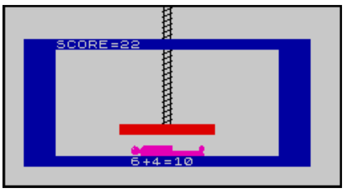
| | |
|---|---|
| **Nápad:** | ■■■■■■■□□□ |
| **Hratelnost:** | ■■■■■■■■■□ |
| **Grafika:** | ■■■■■■■□□□ |
| **Zvuk:** | ■■■■■■■■■■ |
| **Vedikt:** | Je to prostě klasika |

# Crusher

© 198* UTS

**Pokud si myslíte, že vás na základce naučili počítat, mýlíte se. Dr. Death ze hry Crusher vám to rád dokáže.**

BASICovská klasika, kterou se budeme zabývat dnes, je od jisté společnosti UTS. Po té v dnešní době není ani stopy, ale proslýchá se, že ve zlaté éře ZXS skupovala dílka od různých programátorů a platila £10 za kus. Zřejmě je hříčka Crusher jednou z nich.Po spuštění hry se dozvíte, že jste byli polapeni doktorem Smrťákem (Dr. Death) a umístěni do jeho drtiče. Zvrhlý doktor (zjevně zakomplexovaný matematik) vám klade zákeřné matematické otázky, na které musíte rychle (dle zvolené obtížnosti) odpovídat: pravdivost výpočtu




potvrzujete stiskem "1" a dementujete stlačením "0". Pokud se seknete (anebo vám odpověď trvá příliš dlouho), drtič se o stupínek přiblíží k vašemu tělu.Celá tato prima hříčka byla napsána v ZX-BASICu. Za pětikilo bych ji pro někoho v klidu udělal!

**-BLS-**

| | |
|---|---|
| **Nápad:** | ■■□□□□□□□□ |
| **Hratelnost:** | ■□□□□□□□□□ |
| **Grafika:** | ■□□□□□□□□□ |
| **Zvuk:** | ■□□□□□□□□□ |
| **Vedikt:** | Super basicovská hloupost. |

PROGRAMOVÁNÍ

# Bojte se +3 packu!

**Předem bych rád upozornil, že jsem kvák cracknul a ještě větší kvák za +3 packoval, ale z čistě teoretického hlediska jsem nevydržel a ozývám se. Aby se v mých hlodech dalo orientovat, uspořádám je zcela chaoticky, jinak to bohužel neumím, ale alespoň se pocvičíte v crackování nepřehledného sloupku; zkusím to sice srovnat podle pořadí, v jakém reaguji na článek T.D.M., ale známe své pápnhajmské...**

**Problém 1:** Na +2A funguje prý POKE 23388,22. Bodejť ne, POKE s rozumnými parametry funguje všade, otázka je, co provede. Nějaký efekt bude mít leda tak u nějakého šíleného tapaře, který používá 128 Basic, móóóóóžná na BetaShitu ve 128 módu... Většina (a dá se polemizovat, zda ta rozumnější) ale používá spíš něco podobného USR 0 nebo alespoň BS-ROM nebo D40 (či naopak). Nějaký připrdlý POKE pak bude mít za následek jen svinec v paměti a stránkování nulové. Nechápu, proč T.D.M. tvrdí, že na tento poke na +3 můžeme zapomenout, když vlastně neukáže jedinou rutinu, ve které by ho nevyžadoval!
Opravdový +3pack: zdá se, že stránka 7 (T.D.M. píše 23, ale to je snad to samé, my, co jsme četli Lovce mamutů (a zanechalo to v nás hluboký dojem, viz George K.), víme, že vše, co je více jak 5, je MOC, a Huňáč a +GAMA se navíc vyznačují zvláštní schopností počítat až do 10) nemusí být rozpakována jako úplně poslední, stačí, když bude v posledním bloku. To je celkem příjemné (zvlášť, když při pakování +3diskaře zohledním a ono mě to ani moc neomezuje). Házet tam ale i stránku 1, no prosím, ale co když ji při depaku prvního bloku používám jako buffer (to by sice bylo hloupé, ale pak bez ohledu na to, že jsem ji natáhl až v posledním bloku, by byl RamDisk přepsatý)...
Adresa 25000 jako začáteční je velmi hezká, bojím se ale, že často je potřeba trochu nižší (ovšem nižší než 24500 nemám rád). Faktem je, že dosud žijí a fungují crackeři (majitelé D40), kamarádi Universuma nebo alespoň Jirky Vondráčka, co používají hnusnou 24200 (ale ty nelze brát moc vážně, oni například na pakování používají PackMaker), ona je sice výhodná proto, že je dost nízko, v případě, že je něco i pod ní, není třeba pamětí moc hýbat a volné místo se nahoře rovněž ušetří spíš, než u bloků kratších, ovšem podle mne to necelé kilo ani moc nehraje roli. 25000 bych tedy podporoval (a hned začnu přepakovávat všechny Chequered Flagy a jiné píp (cenzura je všude!! pozn. editora.), co začínají na 24000 nebo 24200). Ovšem systém, který posouvá basic jenom proto, aby si někam nacpal svých pár kriploidních systémových proměnných (když má navíc třeba 8kilovou ROMku a do těch druhých 8 kilo (ROM má přece 16, ne?) by se mu vešly sysproměnné i s Basicem minimálně čtyřikát, je poněkud úchylný. Tak to jsem teď jen plkal o ničem a teď to...
**Stránkování**: pokud je pravda, že funguje první způsob, tak se taky bude používat a na ten druhý prosím zapomenout. Proč? Protože každý rozumný člověk má na 23388 prostě nulu z toho důvodu, že tam nemá vlastně vůbec nic, žádnou informaci o ničem, tož nemá smysl si odtud zjišťovat nějaké hodnoty a navíc získaná hausnumera někam OUTovat...
**Přerušení**: proč proboha nechat celou gamesu nebo program běžet s DI? Co když tam někde bude oblíbená instrukce NOP (ta by nevadila, ale je oblíbená) nebo HALT (oblíbená ještě více, ale s DI trochu zlobí, proč asi...)? A proč někam pchát IM 2, když IM 1 stačí? Co u programů, které třeba pod IM 1 čtou klávesnici? Dá se v rámci IM 2 skákat na adresu 56 (nebo kolik to vlastně je...), tedy test klávesnice a posunutí hodin, aniž by se pitomý +3DOS vrátil? Tohle bylo zapomenuto a nevyřčeno. Ale systém, který se chová takhle stupidně, si existenci nezasouloží, a to ani na Spectru...
Proč nelze proboha +3pack spáchat, když nelze nahrát stránku 0 a 7 těsně před spuštěním? O té nule řeč vůbec nebyla a teď je to najednou limitující faktor? No co je to za systém, ten +3DOS? To je závislý na stránce, jejíž obsah je mu v podstatě ukradený? (Nebo je chyba v T.D.M. a přepsal se?)

Proč se namáhat s +3packem: je hezké, že když nám T.D.M. vyblije něco na TAPE, MBčkáři na to nebudou muset šáhnout, ale to bude asi tím, že oni nemusí šáhnout vůbec na nic, co je z TAPE (výjimka - blikavé loadery). Za to, že Matsoft a já nerozeznáme +3fuck od BetaShitu, vůbec nemůžem, protože Pentacle měl mechaniku v krabici od D40 s velkým nápisem +3 (viz Intro v starším ZXM), zatímco u T.D.M. žádný podobný propagační transparent nebyl prezentován. Ovšem proč se s +3packem nenamáhat: pokud bude stránkovat přes 23388 (viz výše) nebo bude odpojovat nějaké připrdlé RamDisky (viz níže), tak mám téměř zaručeno, že u mne fungovat tedy nebude (a dělat jednu verzi pro 4 +3fuckaře, která mi nebude fungovat, a druhou takovou, která šlape mně a oni se z nějakého víceméně banálního důvodu rozčilují, je trápení zvířat a Spectristů).

Špinavý a umytý Maťa: jakýkoliv systém, který považuje normálně užívanou konvenční paměť za svůj RamDisk, je psychicky retardovaný, kriploidní a ouplně stupidní a blbý! Kdyby nějakého 48ičkáře napadlo si někam do memory hodit RamDisk a pak by nás všechny buzeroval s požadavkem, ať mu speciálním způsobem pakujeme hry, že jemu to přepisuje jeho cenný RamDisk, tak by ho asi zabil i T.D.M., nejen já. Přidávat basic nebo rutinu pro jeho zruchání je hovadina (kor když to zase stránkuje přes 23388, takže mi to odOUTuje někam ouplně do tlustého střeva, vím, o čem mluvím, nebo když to hopsá na adresu 319, právě jsem to vyzkoušel, dělá to psí kusy, no prostě hrůza). Prostě RamDisk (používá ho T.D.M. vůbec, zvlášť, když má ten RamDisk tu příjemnou vlastnost, že každým resetem se jeho obsah smaže?) by měl být inaktivován defaultně, nebo, pokud o něj T.D.M. tak moc stojí, by se měl dát inaktivovat před nahrátím hry (cpát to do basicu nebo strojáku hry je blbost - z basicu by to uživatelé stejně mazali (a diví se T.D.M.? Vždyť jim to hroutí počítač!) a stroják by způsobil prostě to, že by hra nefungovala). Měl bych ale jeden (a myslím, že genitální) nápad.

Jak uvolnit 17. stránku pro sebe (já vím, tady jsem něco přehodil, ale říkal jsem, že v tom bude chaos a zmatení): celý ten hnusný strojáček hodit do ROMky (do 48ROMky na +2Ačku), kam, to nevím, nemám totiž tušení, kde tam je volné místo (ne že bych v životě neměl +2Ačko, ale výměna ROMek za nějaké normální bylo to první, co jsem musel učinit (a taky učinil)). No a když už bude v ROMce, bude se volat před nahrátím gamesy. Aby T.D.M. nemusel do her, které si stáhne od nás, nebo dokonce do Maťových +2Apacků přidávat nějaké randomajzy, stačí mu udělat odskok v rutince pro basicový příkaz CLEAR (ten je typický pro začátek nahrávání nějakého progu, a když jako vedlejší efekt zruší RamDisk a tím zajistí, že ten pack bude skutečně +3 a ne +2A, tak tím lépe, ne?). A půjde to krásně bez drastických úprav programů, které by nemajitele +3disku jen vytáčely... Koneckonců, systém, kde se motor mechaniky zastavuje pomocí PAUSE 0 (no to snad T.D.M. nemyslí vážně!) je pořád ještě úchylárna a když může být na vedlejší účel povel PAUSE, proč ne zrovna CLEAR, když je tak děsně šikovný?

...Ještě k té PAUSE, čekat po nahrání na stisk klávesy by mohlo mít smysl tak při nějakém poke ('T'-Training), ale neříkejte mi, že se ten motor prostě nezastaví (jaký je rozdíl mezi vykonáváním kódu povelu PAUSE a prováděním kódu hry?)! Podle všeho se to bez ní krásně obejde (no nevím, nejsem odborník, nemám +3fuck, ale když teď čtu, jaká je to strašná věc, jsem té skutečnosti i rád). Tak a je to. Pokud tam skutečně někdo (úchylný) chce čekání na klávesu, ať si tam hodí Training...

Špinavý a špinavý Klaxon Hollis: já mám sice nejvíc co kecat (myšleno nejmíň, ale nejvíc; to je poznámka pro šťouraly), nicméně do JSH se opřu. Sice řve proti pomalým MrvPackům, ale mnohé jeho depacky nejsou zrovna fast (úplná hrůza jsou třeba Renegade 128). Navíc když při nahrávání problikne obrázek, pak už je obrazovka jen černá a v dálce je tušit depack (jasná Hollisova vizitka, daná tím jeho maniakálním používáním VRAM), samozřejmě co nejpomalejší, je mi líto, ale vypadá to strašně. Úplně nejhorší jsou pak jeho Trainingy - ve spouště her jsou napevno (pak hrajete vždycky už jenom s nekonečnými životy, jako by nešlo nikam dát dotaz (jak znám Freddyho, tak asi nešlo)), ve spouště her na ně není upozorněno (to potom potěší, když se hra konečně rozpakuje a na obrazovce se něco objeví, a jinak klid, to váhání, jestli vůbec něco zmáčknout a jestli zmáčknout T, zda tam vůbec Training je...), ovšem tvrzení, že jeho cracky nejedou na 128, ale jen v USR 0, mne překvapuje (že by to prasa šahalo i na printbuffer?). Nebudu se teď hádat o koncepcích a antikoncepcích či kontracepcích

pakování, tady máme skutečně ještě rezervu - předělat (asi hlavně starší) Freddyho packy na +3 (nebo +2A, on si to T.D.M. už doprasí sám). Že Norové nebo Angličané podporují +3, no prosím (jsou to blázni...), třeba by nebylo od věci +3disk emulovat třeba na MBčku nebo D40 (bylo!!!)..., cpát ale nějaké hloupé nápisy do úvodního basicu hry je ovšem barbařina (jména hloupých...), musím říct, že mne tento nešvar (který zavedl buď Fuka, nebo A.C.G.), dráždí. Teď už jenom znovu zopakuji, že nechápu, proč T.D.M. považuje program, spouštějící se s povoleným přerušením, za pohromu (to bude tím ouchylným systémem; MBdiskaři jako DRON naopak povolené přerušení rádi, protože jim např. hraje AY hudba i během diskových operací - to myslím vážně) a dodám, že vkládat do basicu takového programu POKE 23611,204 je zbytečnost, protože u čtyř lidí v republice to něco udělá a u dvaceti tisíc to jenom bude zabírat paměť (nebude; oni to vymažou...)!

No a to jsou konečně všechny mé stupidní hlody. Užijte si je a radši nic nepakujte, protože se pak budete třást před T.D.M. (proč to není +3pack) nebo před někým jiným (proč to není třeba WafaDrive pack). Bojte se +3packu!

**-+GAMA-**

# Kanály ZX SPECTRA

**Jak připojit tiskárnu k ZX Spectru? Jak pro ni napsat obslužný program ve strojovém kódu? Jak informovat tento program, co má vlastně tisknout? Na otázky tohoto typu dává odpověď následující příspěvek.**

ZX Spectrum je počítač, u kterého je již počítáno s tím, že jeho uživatelé k němu budou chtít připojovat různá periferní zařízení (nejčastěji tiskárnu). Systém, který umožňuje připojení těchto periferií, je naprosto univerzální. Ke Spectru lze totiž připojit tiskárnu s libovolným druhem komunikace. Abychom toto připojení mohli provést, musíme znát, jak pracují kanály a linky našeho počítače. Počítač totiž se svými periferiemi komunikuje právě pomocí těchto kanálů a linek.

Linka je cesta, kterou se posílají jednotlivé zpracovávané znaky z počítače do periferie, nebo jsou periferií vysílány a počítač je přijímá, přičemž typ periferie je označován jako kanál. Potom kanál nemá nic společného s hardwarovou konstrukcí té které periférie. Jedná se o čistě softwarové zajištění komunikace mezi hlavním programem a programy pro obsluhu periferií. Komunikace zde probíhá sériově, po jednom znaku (bytu). Konkrétně to vypadá tak, že chce-li program vyslat nějaký text na periferní zařízení, bere postupně jednotlivé znaky (byty) tohoto textu a předává je určitým způsobem podprogramu, který zpracování znaků zajistí (nejčastěji vysílání na port apod.). Podprogram je volán vždy znovu a znovu při vysílání dalšího znaku. Obrovskou výhodou tohoto uspořádání je fakt, že znaky, které mají být vyslány na periferní zařízení, vůbec nemusejí být uloženy v paměti počítače a tudíž lze pracovat s datovými soubory daleko většími, než operační paměť. Při vstupu z periferie do počítače probíhá komunikace obdobně. Program zavolá vstupní podprogram, který převezme znak z periferního zařízení a předá jej hlavnímu programu.

Nyní se podívejme, jak taková komunikace vypadá u ZX Spectra. Všechny vstupní i výstupní podprogramy jsou samozřejmě napsány ve strojovém kódu (jinak to ani nejde, vezmeme-li v úvahu rychlost ZX Basicu). Znak, který se přijímá nebo vysílá, je uložen v registru A (akumulátoru) mikroprocesoru Z-80. Volání výstupního podprogramu, chceme-li například vytisknout znak "A", může vypadat následovně:

```
PRT1    ld a,"A"  ;znak do akumulátoru
        call outp ;volání výstupní rutiny.

OUTP:             ;vyslání znaku
                  ;na periferní zařízení
        ret       ;návrat do hlavního progra-
                  ;mu
```

Toto je ovšem zjednodušený příklad, neboť řídící program předpokládá, že výstupní podprogram začíná na určité konkrétní adrese, což zdaleka nemusí být splněno (například toto místo již může být obsazeno jiným programem). Proto se tento problém řeší tabulkou, ve které jsou uvedeny počáteční adresy vstupních a výstupních podprogramů a hlavní program při volání potřebného podprogramu vždy bere patřičnou adresu z této tabulky:

```
PRT2    ld a,"A"        ;znak do
                        ;akumulátoru
        ld hl,(tab)     ;adresa rutiny do
                        ;HL
        ld de,CONT      ;adresa
                        ;pokračování
        push de         ;programu do
                        :zásobníku.
        jp (hl)         ;zavolání rutiny.
CONT                    ;zde pokračuje
                        :hlavní program.

TAB     defw OUTP       ;adresy všech výs
                        ;tupních
        defw OUTP2      ;rutin
        defw OUTP3

OUTP                    ;vyslání znaku
                        ;na periferní
                        ;zařízení.
        ret             ;návrat.
```

Nyní jsme vyřešili případ, že výstupní podprogram může být uložen na libovolném místě v paměti a jeho napojení na systém se provede zapsáním jeho počáteční adresy do patřičného místa tabulky. Potíže nastanou tehdy, jestliže tato tabulka nebude souvislá, ale rozdělená na více částí, mezi kterými budou například umístěny vstupní a výstupní obslužné podprogramy. Pak musíme přidat ještě další tabulku, ve které budou odkazy na patřičná místa původní tabulky s adresami obslužných podprogramů (dvojnásobná nepřímá adresace). Celá tato konstrukce vypadá na první pohled dosti komplikovaně, proto její činnost napřed vysvětlím na funkci ZX Basicu.

ZX Spectrum má šestnáct linek číslovaných od nuly do patnácti. Každá z těchto linek může být zároveň vstupní i výstupní. Máme-li samostatný počítač (tj. bez připojeného ZX Interface 1) tak tento počítač sám o sobě již používá dva kanály a sice klávesnici a pole obrazovky (ty jsou sice součástí Spectra, ale komunikace s nimi probíhá stejně jako s "vnějšími" periferiemi).

Třetím kanálem může být ZX Printer, je-li připojen. Majitelé ZX Interface 1 navíc mohou používat sériový výstup a vstup RS 232, network a microdrive. Jak se s těmito kanály pracuje?

Pro připojení některého kanálu k lince slouží příkaz OPEN. Uvedu příklad:

```
OPEN #5,"s"
```

připojí pátou linku na obrazovku. Kanály se u Spectra označují jednotlivými písmeny podle tabulky:

| | | |
|---|---|---|
| klávesnice | "k" | (keyboard) |
| obrazovka | "s" | (screen) |
| ZX Printer | "p" | |
| RS 232 | "t" nebo "b" (text, binary) | |
| network | "n" | |
| microdrive | "m" | |

Kanály "t", "b", "n" a "m" se týkají jen příkazů vyžadujících ZX Interface1. Příkaz PRINT vždy vysílá znaky do linky dvě, zatímco příkaz INPUT bere znaky standardně z linky jedna. Lze však tyto příkazy přesměrovat i na jiné linky. Např.:

```
PRINT #5;"Nazdar"
```

vytiskne text na obrazovce (jestliže jsme předtím samozřejmě provedli OPEN 5,"s"). Jestliže ovšem napíšeme

```
OPEN #5,"p": PRINT #5;"Nazdar"
```

vytiskne se text na tiskárně. Totéž nastane při

```
PRINT #3;"Nazdar"
```

z čehož vidíme, že příkaz LPRINT je v podstatě totéž jako PRINT #3. Podobně PRINT je totéž jako LPRINT #2. Příkaz INPUT lze také psát ve formě

```
INPUT #1;a
```

ale to u samotného Spectra (bez Interface 1) nemá význam, protože z tiskárny nebo obrazovky stejně nelze číst. Dalším příkazem pro práci s linkami je CLOSE. Slouží k uzavření dané linky (odpojení vstupní nebo výstupní rutiny). Jestliže provedeme

```
CLOSE #5: PRINT #5;"Nazdar"
```

způsobí to chybové hlášení (Invalid I/O device).

Programy ve strojovém kódu komunikují s periferiemi také pomocí linek, používají dokonce stejné linky jako ZX Basic. Jde jen o to, jak si mají potřebnou linku otevřít a používat ji. V systémových proměnných Spectra je proměnná STRMS (start 23568, délka 38 bytů). V této proměnné jsou uloženy informace o všech

linkách. Proměnná je organizovaná jako devatenáct (místo šestnácti) dvoubytových hodnot příslušejících jednotlivým linkám. Proč je jich devatenáct místo šestnácti? První tři hodnoty patří totiž linkám mínus tři až mínus jedna, které používá Basic (editor, tisk chybových hlášení a reportů „Start tape…„ atd.), které nejsou ani obvyklými příkazy přístupné a do kterých není dobré zasahovat pomocí příkazů POKE ani jinak. Čtvrtá až devatenáctá hodnota udávají relativní posunutí (rozumí se od začátku) v tabulce obsahující adresy vstupních a výstupních (dále jen I/O) rutin představujících obsluhu jednotlivých kanálů. Tato důležitá tabulka začíná na adrese, která je obsahem systémové proměnné CHANS (adresa 23631). Je-li některá hodnota v proměnné STRMS nulová, je daná linka uzavřená (nevede na žádný kanál). Je-li linka otevřená, patřičná položka v STRMS je rovna alespoň jedné. Pro lepší pochopení následuje výpis těchto proměnných tak, jak jsou inicializovány po zapnutí počítače:

```
STRMS     defw 1, 6, 11        ;linky -3 az -1.
          defw 1               ;linka 0.
          defw 1               ;linka 1.
          defw 6               ;linka 2.
          defw 16              ;linka 3.
          defw 0, 0, 0,…0      ;ostaní uzavřené.

CHANS     defw CHAN_AD         ;adresa tabulky
                               ;kanálů.
                               ;tabulka kanálů.
CHAN_AD defw #09F4             ;adresa výstupní a
          defw #10A8           ;vstupní rutiny
          defb "K"             ;pro klávesnici.

          defw #09F4           ;adresy rutin
          defw #15C4           ;pro obrazovku.
          defw "S"
          defw #0F81           ;adresy rutin
          defw #15C4           ;pro Basic editor.
          defb "R"
          defw #09F4           ;adresy rutin
          defw #15C4           ;pro ZX Printer
          defb "P"
```

Tabulka kanálů na adrese (CHANS) je organizována jako čtyři až šestnáct pětibytových položek (podle toho, kolik kanálů je vytvořeno). Každých pět bytů obsahuje adresu výstupní rutiny (první dva byty), dále adresu vstupní rutiny (druhé dva byty) a písmeno (pátý byte). Písmeno udává typ kanálu, tj. "k", "s", "p" atd. Sedmý bit pátého bytu může být nastaven nebo zrušen podle toho, zda daný kanál je vytvořen trvale nebo přechodně. Toto nás nemusí zajímat, protože pro nás jsou důležité jen trvalé kanály (s resetovaným sedmým bitem). Přechodné kanály používá ZX Interface 1 pro příkazy MOVE, SAVE*, LOAD*, MERGE* a VERIFY*. Ukazatel do tabulky kanálů dostaneme jako součet obsahu proměnné CHANS a obsahu patřičné položky proměnné STRMS, přičemž takto vzniklé číslo je adresou druhého bytu v pětici (protože nulová hodnota položky v STRMS není pro otevřenou linku možná, pointer ukazuje na druhý byte). Adresu I/O rutiny musíme bohužel získávat takovýmto složitým způsobem. Druhá tabulka (obsahující seznam vytvořených kanálů) není totiž souvislá, poněvadž mezi jednotlivými péticemi bytů mohou být ještě vyrovnávací paměti (buffery) pro network a microdrive.

Naštěstí můžeme volat také rutiny z ROM, které za nás hodně zařídí (abychom nemuseli počítat a hledat adresu I/O rutiny při vysílání každého znaku). K dispozici jsou celkem tři, které budeme potřebovat. Především je to rutina CHAN_OPEN (adresa #1601), která umožní směrovat vstup a výstup následujících znaků na určitou linku (která ovšem musí být otevřena). Číslo aktualizované linky uložíme při volání rutiny do akumulátoru. Rutina v podstatě provede to, že uloží adresu v tabulce kanálů CHANS do systémové proměnné CURCHL (adresa 23633), aby se vždy nemusela vypočítávat. Máme-li takto aktualizovaný kanál, můžeme na něj vysílat nebo z něj přijímat znaky. Vysílání znaku provede rutina PRINT_CHARAKTER (na adrese 0010), která se dá volat instrukcí RST #10.

Kód znaku je při volání rutiny v akumulátoru. Chceme-li naopak znak číst, zavoláme rutinu INPUT_AD (na adrese #15E6). Rutina uloží přečtený znak do akumulátoru a je-li tento znak platný, nastaví příznak přenosu C. Rutiny PRINT_CHARACTER a INPUT_AD používají na začátku a konci instrukci EXX, takže nemusíte při jejich volání uschovávat aktuální sadu registrů do zásobníku. Pro názornost nyní uvedu příklad, jak bude vypadat již známý text na obrazovce, tentokrát však pomocí strojového kódu:

```
end        equ #FF         ;zarázka.

PRT_PXT    ld a,#02        ;číslo kanálu do a.
```

```
         call CHAN_OPEN    ;aktualizace
                           ;kanálu.
         ld hl, TEXT       ;adresa textu do
                           ;HL.
LOOP     ld a,(hl)         ;znak do A.
         cp END            ;test na zarázku.
         ret z             ;návrat při
                           ;zarázce.
         rst PRINT_CHARAKTER
                           ;vysílání znaku na
                           ;"s".
         inc hl            ;pointer na další
                           ;znak.
         jr loop           ;opakuj pro další
                           ;znak.
TEXT     defm "Nazdar"     ;zobrazovaný text.
         defb END          ;konec textu.
```

Teď už jde jen o to, jak napojit na stávající kanálový systém naši I/O rutinu. Je potřeba uložit její adresu na patřičné místo v tabulce kanálů (CHANS). Princip je jasný, uvedu zde proto program pro napojení výstupní rutiny pro tiskárnu na třetí linku (ta je tiskárně standardně vyhrazena):

```
OPEN_P   ld hl,(2*6+STRMS) ;posunutí od
                           ;CHANS do HL.
         ld de, (CHANS)    ;(CHANS) do DE.
         add hl,de         ;adresa v tabulce.
                           ;CHANS v HL..
         push hl           ;HL do zásobníku.
         ld hl,OUTP-OPEN_P;rozdíl adres
                           ;do HL.
         add hl,bc         ;adresa OUTP v
                           ;HL.
         pop de            ;adresa v CHANS
                           ;v DE.
         ex de,hl          ;záměna DE a HL.
         ld (hl),d         ;ulození adresy
                           ;OUTP
         dec hl            ;do tabulky v
                           ;CHANS.
         ld (hl),e
         ret               ;návrat.

         OUTP              ;vysílání znaku
                           ;na periferní
                           ;zařízení.
         ret               ;návrat.
```

Adresa rutiny OUTP je určena jako součet relativního posunutí od adresy OPEN_P a obsahu registru BC. Při volání funkce USR z Basicu je totiž adresa této rutiny uložena do BC.
Ještě několik poznámek k umístění I/O rutin do paměti Spectra. Rutiny lze samozřejmě umístit do paměti kamkoliv. Nejlepší je ovšem umístění do printer-bufferu od adresy 23296, protože nepoužívá-li se ZX Printer, tato oblast je volná. Jestliže se sem rutina nevejde, dáme ji někam nad RAMTOP (alespoň adresa posledního CLEARu plus jedna). Zásadně nedoporučuji umísťovat rutiny do řádek Basicu nebo do nějaké proměnné vytvořené příkazem DIM. Je to sice pohodlné, nicméně to způsobuje "záhadné" hroucení systému při posunutí Basicového programu na jiné adresy, než si myslíme (program v Basicu totiž nemusí začínat na adrese 23755, jak se mnozí majitelé Spectra mylně domnívají).
Používání linek a kanálů na Spectru má velké výhody. Nejde zdaleka jen o připojování tiskáren, ale např. na disketové jednotce DIDAKTIK D40/D80, využijeme znalostí kolem kanálů při práci se sekvenčními soubory. Z Turbo assembleru PROMETHEUS můžeme tisknout zdrojové texty přes kanál 3, pokud na tento kanál připojíme sekvenční soubor, dostaneme zdrojové texty například do DESKTOPU. Používáním kanálů se vyhneme různým těžkostem při využívání datových souborů vytvořených na jiném počítači (můžeme pracovat se souborem mnohem větším než je operační paměť počítače.
Pro všechny Spectromaniaky sepsal

**-Jiří Lamač-**

# Zázraky v BASICu

***lekcia 07***

**Rekurzívne funkcie pomocou DEF FN. Čo myslíte, dajú sa pomocou DEF FN definovať aj rekurzívne funkcie ? Iste ste ľahko uhádli, že sa samozrejme dajú, veď inak by som vám túto otázku nedával...**

Pozrime sa na štruktúru takej rekurzívnej funkcie. Na začiatok si vezmime nejaký jednoduchý príklad - výpočet faktoriál. Faktoriály sú definované
takto:
N! = N*(N-1)*(N-2)*...*2*1. Čiže slovensky povedané faktoriál čísla N je súčin všetkých celých kladných čísel od 1 po N. Túto funkciu by sme mohli vypočítať nejak takto:

N! = IF (N=1) THEN 1 : REM nerekurzívna vetva
IF (N>1) THEN N * (N-1)! : REM rekurzívna vetva

Ibaže toto je síce napísané veľmi pekne, ale do DEF FN sa to zapídsať nedá, pretože vo výraze nemôžeme použiť príkaz IF-THEN. Avšak môžeme použiť akúsi vzdialenú náhradu za IF-THEN - a tou náhradou je binárny operátor AND. Tento operátor pracuje nasledovne:

X AND Y vráti: X, ak je Y nenulové (X môže byť aj číslo aj reťazec) 0, ak je Y nulové a X je číslo prázdy reťazec ak je Y nulové a X je reťazec.
Toto sa dá veľmi pekne využiť pri náhrade IF-THEN vo funkciách definovaných "vidličkou". To sú také funkcie, ktoré majú viac definícii na rôznych intervaloch. Napríklad absolútna hodnota by sa dala definovať takto:

ABS X = IF (X>=0) THEN X
IF (X<0) THEN -X

Keď to zapíšeme do DEF FN a nahradíme IF-THEN operátorom AND, bude to vyzerať takto:

DEF FN a(x)=(x AND x>0)-(x AND x<0)

(x<0) je výraz, ktorý nadobúda hodnotu 0 (akože nepravda) ak je X nezáporné.

Skúsme si teraz napísať nejakú inú funkciu, definovanú "vidličkou":

Funkcia B(X) = IF X<=0 THEN X^2
IF X>0 THEN LN X
DEF FN b(x)=(x*x AND x <= 0)+(LN x AND x>0)

Lenže tento zápis pomocou DEF FN má jednu velikú slabosť: keď dáte počítaču vypočítať koľko je FN b(-2) tak z toho zblbne a vypíše chybové hlásenie "Invalid argument". Prečo ? Operátor AND totiž vyhodnocuje prvý oprerand aj v prípade, že druhý je nulový. To znamená že funkcia LN sa vyhodnocuje aj vtedy ak X je záporné alebo nulové.
Za každú cenu musíme zabepečiť aby sa v prípade X<=0 nevyhodnocovala funkcia LN. Keďže pri vyhodnocovaní aritmetického výrazu sa vždy vyhodnocujú všetky členy tohto výrazu, nemôžeme týmto spôsobom písať rekurzívne funkcie. Pri vyhodnocovaní výrazu by sa nám zakaždým začala vyhodnocovať aj rekurzívna vetva funkcie, čo by viedlo k zacykleniu a preplneniu zásobníka a skončilo by to chybovým hlásením (prípadne kolapsom celého systému)...

Aby sme tomu zabránili, musíme použiť nejaký "špinavý" trik. Príklad takého "špinavého" triku je použitie funkcie VAL "reťazec" spolu s operátorom "reťazec" AND podmienka. Spravíme jednu veľmi jednoduchú vec: Akonáhle bude X kladné, tak funkcii VAL podhodíme na vyhodnotenie reťazec "LN x" a v opačnom prípade jej podhodíme reťazec "x*x". Naša "vidličková" funkcia bude potom vyzerať takto:

DEF FN c(x)=VAL (("x*x" AND x <= 0)+("LN x" AND x>0))

Týmto pádom sme zabezpečili aby sa nevyhodnovcovalo LN X pre X<=0. Keď si namiesto LN predstavíme rekurzívnu vetvu nejakej funkcie, môžeme týmto spôsobom veľmi pekne písať aj rekurzívne funkcie. Napríklad naše faktoriály zapísané rekurzívne budú vyzerať takto:

DEF FN f(n)=VAL (("1" AND n<2)+("n*FN f(n-1)" AND n >= 2))

Teraz trochu odbočme od čistej matematiky a skúsme si napísať jednu zaujímavú funkciu - nazvime ju "násobenie reťazca číslom". Na vstupe bude mať reťazet A$ a číslo N, na výstupe bude dávať reťazec pozostávajúci z N bezprostredne po sebe idúcich reťazcov A$.
Príklad:
Nech A$="zx" a N=4 potom vysledok bude "zxzxzxzx". DEF FN m$(a$,a)=VAL$ (("""""" AND a<1)+("a$+FN m$(a$,a-1)" AND a >= 1))

Všimnite si že aj táto funkcia je písaná rekurzívne. Tých šesť úvodzoviek za sebou predstavuje reťazec pozostavajúci z dvoch znakov - dvoch uvodzoviek.
Práve šesť ich je preto lebo ak sa majú v reťazci vyskytovať uvodzovky, treba ich napísať dvakrát.
Použitie tejto funkcie je veľmi rozmanité. Napríklad chceme vypísať na obrazovku N znakov '#'. Namiesto klasického:

FOR a=1 TO N: PRINT "#";: NEXT a
použijeme oveľa elegantnejšie
PRINT FN m$("#",N);

Alebo chceme zarovnávať vypisované čísla podľa pravého okraja. Dá sa to spraviť takto:

PRINT "Hodnota:"; TAB (31-LEN STR$ x);x

Lenže keď chceme aby medzi nápisom "Hodnota" a samotným číslom boli napr. bodky, môžeme s výhodou použiť našu funkciu takto:

PRINT "Hodnota";FN m$(".",23-LEN STR$ x);x.

Na záver si spravme niečo zložitejšie. Skúsme si definovať funkciu "reverse" - obrátenie poradia znakov v reťazci. Ak bude na vstupe napríklad reťazec "abcd", potom na výstupe bude "dcba". Nech táto naša funkcia pracuje takto: Vezme posledný znak reťazca a za neho dopíše zvyšok reťazca, ktorý medzi tým už prešiel našou funkciou "reverse". V prípade reťazca dlhého maximálne jeden znak nič netreba obracať a v tom prípade sa vráti ten istý reťazec.

DEF FN r$(a$)=VAL$ (("a$(LEN a$)+FN r$(a$( TO LEN a$-1))" AND LEN a$>1)+("a$" AND LEN a$ <= 1))

Funkcia by sa dala definovať ešte druhým spôsobom: vezmeme prvý znak reťazca a tento prvý znak zapíšeme za "reverznutý" zvyšok reťazca. Zápis takto definovanej funkcie bude vyzerať takto:

DEF FN o$(a$)=VAL$ (("FN o$(a$(2 TO ))+a$(1)" AND LEN a$>1)+("a$" AND LEN a$ <= 1))

Na prvý pohľad je všetko v poriadku a aj tento druhý spôsob zápisu by mal bez problémov fungovať. Ale keď si ho vyskúšate, zistíte že akosi predsa len nefunguje, hoci v ňom nie je žiadna do očí bijúca chyba. V čom je ten povestný háčik, prečo to nefunguje tak ako by malo, nechávam už na vašu vlastnú úvahu a bádanie. Pre znalcov systému iste nebude žiadny problém odhaliť, v kde je pes zakopaný...

**-BUSY-**

# Strojový kód pre pokročilých

*lekcia 09*

**V predchádzajúcej lekcii sme riešili problém ako môžeme z hodnoty čísla vytvoriť postupnosť desiatkových číslic, ktoré zodpovedajú tomuto číslu. V tejto lekcii budeme riešiť ten istý problém, ale namiesto desiatkovej sústavy budeme pracovať v šestnástkovej - budeme vytvárať postupnosť šestnástkových číslic.**

Na riešenie tohto problému použijeme ten istý program, len si ho prispôsobíme na potreby šestnástkovej sústavy. Toto prispôsobene vykonáme tromi nasledovnými zmenami:

**1.** Náš nový program bude vytvárať číslo v šestnástkovej sústave, preto namiesto mocnín čísla 10 v ňom budú mocniny čísla 16.

**2.** Na vyjadrenie čisel ktoré možno zobraziť v párovom registri nám v šestnástkovej sústave stačia iba štyri číslice namiesto piatich v desiatkovej sústave. Preto bude modifikovaný program pracovať iba nad týmito štyrmi číslicami.

**3.** ASCII-kódy šestnástkových číslic na rozdiel od ASCII-kódov desiatkových číslic nejdú bezprostredne za sebou, ale sú rozdelené na dve skupiny - číslice 0-9 a písmená A-F. Kódy v jednotlivých skupinách už idú bezprostredne za sebou. Toto budeme musieť vhodným spôsobom v programe ošetriť. Náš program z minulej lekcie po týchto troch úpravách bude vyzerať takto:

```
run     ld hl,#fc12      ;Príklad čísla
bnhx    ld bc,buffer     ;Adresa buffera
                         ;pre číslice
        ld de,-#1000     ;pre  šestnástkový
                         ;rád 4096 (16^3)
        call cifra       ;výpočet číslice
        ld de,-#0100     ;..256-ky (16^2)
        call cifra       ;výpočet číslice
        ld de,-#10       ;šestnástky (16^1)
        call cifra       ;výpočet číslice
        ld e,-#01        ;...jednotky (16^0)
cifra   ld a,'0'-1       ;výpočet číslice
                         ;jedného rádu
cif1    add hl,de        ;odpočítavame
                         ;hodnotu radu
        inc a            ;(súčasne počí
                         ;tame kolko krát)
```

```
        jr c,cif1        ;az kým
                         ;neprekročíme 0
        sbc hl,de        ;Oprava prekroče-
                         ;nia nuly
        cp '9'+1         ;číslica väčšia
                         ;ako 9 ?
        jr c,cif2        ;nie - skok
        add a,'a'-':'    ;áno-korekcia
                         ;číslic A-F
cif2    ld (bc),a        ;ulozenie číslice
                         ;do buffera
        inc bc           ;ukazovateľ na
                         ;ďalší miesto v bu
        ret
buffer  db 'xxxxxxxx'    ;buffer pre
                         ;naše číslice
```

To, čo sme práve urobili, bola iba modifikácia už existujúceho riešenia z minulej lekcie. Je to dobrý a v praxi veľmi používaný postup, ale tento postup nám neprináša žiadne väčšie originálne myšlienky. Skúsme sa preto teraz pozrieť na tento problém z úplne inej strany! Na vyjadrenie jednej šestnástkovej číslice potrebujeme práve štyri bity (pretože 2^4=16). Párový register má 16 bitov (rozdeľme si to na štyri štvorice bitov) a číslo v ňom vyjadrené sa dá zapísať štyrmi šestnástkovými číslicami. Keď sa nad tým zamyslíme, prídeme na to, že jednu konkrétnu šestnástkovú číslicu určuje iba jej prislúchajúca štvorica bitov v párovom registri. Nedala by sa táto vlastnosť vhodne využiť v náš prospech?

Vytvorme si rutinku (nazvime ju "bnhx4") ktorá nám zo štyroch bitov (napr. štyri najnižšie bity v akumulátore) vypočíta ASCII-kód príslušnej šestnástkovej číslice a uloží ho do buffera. Týmto sa celý problém prevodu čísla na postupnosť šestnástkových číslic redukuje na postupné aplikovanie rutinky "bnhx4" na všetky štvorice bitov v našom čísle - v poradí od najvýznamnejšej štvorice po najmenej významnú. Teraz si vytvorme rutinku "bnhx8" ktorá vezme jeden bajt (dve štvorice bitov v akumulátore) a aplikuje rutinu "bnhx4" najprvna vyššie a potom na nižšie štyri bity tohto bajtu. Nakoniec vytvorme poslednú rutinku "bnhx" ktorá vezme naše číslo v (párovom registri HL) a aplikuje rutinku "bnhx8" najprv na register H (vyšši bajt) a potom na register L (nižší bajt).

```
run     ld hl,#fc12      ;Príklad čísla
bnhx    ld bc,buffer     ;Adresa buffera
                         ;pre číslice
        ld a,h           ;vyšší bajt = rády
                         ;4096 a 256
        call bnhx8       ;spracovanie
                         ;jedného bajtu
                         ;čísla
        ld a,l           ;nižší bajt = rády
                         ;16 a 1
bnhx8   push af          ;úschova jedného
                         ;bajtu čísla
        rrca             ;presun vyšších
                         ;štyroch bitov
        rrca             ; tohto bajtu
        rrca             ;do nižších bitov
        rrca             ;kvoli výpisu
        call bnhx4       ;spracovaní týchto
                         ;presunutých bitov
        pop af           ;obnova pôvod-
                         ;ného bajtu čísla
bnhx4   and #0f          ;vynulovanie
                         ;nezaújímavých
                         ;bitov
        add a,'0'        ;Výpočet ASCII-
                         ;kódu číslic 0-9
        cp '9'+1        ;číslica větší ako 9?
        jr c,bnhx0       ;nie-skok
        add a,'a'-':'    ;áno-korekcia
                         ;číslic A-F
bnhx0   ld (bc),a        ;ulozenie číslice
                         ;do buffera
        inc bc           ;ukazovateľ na
                         ;ďalšie miesto v
                         ;bufferi
        ret
buffer  db 'xxxxxxxxx'   ;buffer pre naše
                         ;číslice
```

Na domácu úlohu sa skúste zahrať na samotný procesor Z80 a skúste si tento program prejsť od začiatku (návestie "run") až do samotného konca. Práve takto najlepšie uvidíte a na vlastnej koži pocítite, kadiaľ a koľkokrát musí procesor prebehnúť aby mohol splniť úlohu, ktorou sme ho poverili. Zvlášť si dávajte pozor na zásobník - čo a kedy sa doň ukladá a čo a kedy sa z neho číta.

**-BUSY-**



*naši spokojení čtenáři*

# MultiTech

## ...jak na to?

*lekce 04*

**Posledně jsme se zabývali zobrazováním v režimu MultiColor po celé obrazovce. Nadešel čas se naučit něco o konverzi do MultiTechu. Naším cílem je předlohu (fotografii, obrázek atd.) "dostat" do ZXS v co nejvyšší kvalitě.**

Začněme ideálním příkladem. V PC si připravíme grafický soubor (BMP, JPG, GIF...). Za použití nějakého grafického programu (např. PaintShop Pro-můžete si jej stáhnout z www.jasc.com) jej převedeme do palety 256 odstínů šedi (Greyscale). Podle potřeby spravíme jas/kontrast/gamma korekci tak, aby v obrázku byly vyváženě zastoupeny pokud možno všechny odstíny palety od černé po bílou-zde je třeba experimentovat. Jakmile jste spokojeni s podobou obrázku, přizpůsobte jeho rozměry na velikost 256x192 (příkaz Resize). Vznikne vám tak předloha, kterou nyní převedeme do ZXS. Obrázek uložíme na disketu jako .raw, díky čemuž nám vznikne soubor o 49152 bajtech (256 pixlů horizontálně x 192 pixlů vertikálně, co pixel do bajt). Jeho struktura je velmi jednoduchá; soubor .raw nemá žádnou hlavičku, každý bajt souboru odpovídá příslušnému pixelu obrázku (1. bajt popisuje intenzitu pixelu vlevo nahoře, 256. vpravo nahoře, 257. vlevo druhý pixel z vrchu, 49152. vpravo dole). Bajt 0 říká, že pixel je černý, 127 je 50% šeď, 255 je ekvivalentem bílé.

Nyní je na vás, jakým způsobem převedete připravený soubor z PC diskety do ZXS. Každý určitě zná nějaký způsob, jak si zpřístupnit libovolný soubor na PC disketě (majitelům MB-02+ doporučuji použít konverzní program DiscoBolos). Cílem je soubor s obrázkem převést na ZXS disketu. Jakmile jsme hotovi, grafický soubor si načteme do paměti tak, abychom jej měli k dispozici (majitelům ZXS 128 doporučuji soubor otevřít do volných paměťových stránek-každých 16K do jedné, 48mičkáři budou muset soubor otevírat natřikrát-napřed načtěte prvních 16K na adresu 49152 a proveďte konverzi do MultiTechu (popis následuje) a totéž proveďte se zbývajícími 32K souboru). Tak, doufám, že jste všichni hotovi a máte obrázek nějakým způsobem načtený v paměti ZXS. Můžeme tedy přistoupit ke kroku nejdůležitějšímu, totiž samotné technice konverze.

Obě metody, které si dnes popíšeme, jsou postaveny na myšlence, že ZXS má 15 různých barev, které zobrazeny na černobílé TV (či barevné se staženými barvami) vytvoří více či méně plynule přecházející paletu šedi. Pro jednoduchost budeme v našich příkladech využívat pouze osmi barev (mějme na paměti, že zbývajících 7 jsou pouze vyjasněnými variantami původních barev a nedají se nastavit samostatně, tedy beze změny barvy celého atributu).

### Metoda MultiTech HalfPix

Metoda HalfPix je postavena na myšlence, že jeden miniatribut (pod tímto termínem si představ jednu osminku klasického atributu-viz předchozí lekce) je možno rozdělit na dva pseudopixely: jeden z nich je tvořen čtyřmi nastavenými body (bity 7, 6, 5, 4) a barvou inkoustu, druhý má zbývající bity (3, 2, 1, 0) vynulovány a jsou obarveny barvou papíru. Pak stačí si z původní předlohy vytáhnout skupinku vždy 4 bajtů, ty sečíst a vydělit čtyřmi, čímž získáme odpovídající obarvení příslušných 4 pixelů (nezapomeňte ovšem snížit barevnou hloubku takto získaného bajtu-mějte na paměti, že barvy původní předlohy jsou z palety 256 odstínů, tzn., že výpočtem získaný bajt musíme vydělit 32, získáme tak hodnotu 0-7 odpovídající příslušné barvě ZXS (černá až bílá) a teprve tu použijeme na příslušnou složku (ink/paper miniatributu)).

Tato technika nevyniká kvalitou výsledného obrázku, oproti technice popsané dále je ovšem o 50% úspornější-zabírá 6144 bajtů (s tím, že je ovšem třeba pixelovou část Video RAM zaplnit bajtem %11110000).

### Metoda MultiTech 7216

Implementace této techniky je pravda kapánek složitější, ale její výsledky skutečně stojí za to. Zjednodušeně řečeno se jedná o zdokonalenou

techniku HalfPix ovšem s tím rozdílem, že bity v pixelové části Video RAM přizpůsobíme tak, aby co nejvíce odpovídaly původní předloze z PC. Nabízí se několik variant; první, která nás napadne spočívá ve vyhledání 2 nejčetnějších barev vždy ve skupince osmi originální předlohy a následném zkonstruování miniatributu a pixelové části. Můžete si zkusit tuto metodu implementovat-zjistíte, že nepřináší kdovíjak dobré výsledky. Je to jasné-předpokládejme, že v původní předloze najdeme tuto kombinaci 8 barevných pixlů: 6 6 6 0 0 7 7 7
Kdybychom ke konstrukci použili 2 nejčetnější barvy, skončili bychom u žluté (6) a bílé (7), ale úplně by se nám vytratila nejvýraznější složka tohoto motivu-černá (0). Proto by ideální kombinace bodů v tomto příkladu po konverzi do ZXS formátu vypadala takto:7 7 7 0 0 7 7 7 (všimněte si, že v tomto případě se žlutá (6) přetransformovala v bílou (7). Ptáte-li se, jak tedy algoritmizovat převod, aniž byste museli vše manuálně dokreslovat, čtěte dál.

**1)** Z původní PC předlohy si načteme 8 bajtů (každý z nich reprezentuje svou hodnotou 0-255 jeden z osmi pixelů).
**2)** Bajty si (např. bublinkáčem) setřídíme vzestupně.
**3)** Všech osm bajtů sečteme a vydělíme osmi, získáme si tak jejich průměr (dále hodnota A).
**4)** Začneme konstruovat pixelovou část VideoRAM (adresa 16384):

- -vezmeme si 1. bajt z osmi
- -srovnáme s A
- -je-li bajt vyšší nebo rovný A, nastavíme bit 7 adresy 16384
- -je-li bajt menší než A, vynulujeme bit 7 adresy 16384
- -takto postupujeme, dokud nezpracujeme všech osm bajtů (a tedy i bitů 7-0 adresy 16384)

**5)** Nyní zkonstruujeme miniatribut: 1. bajt z posloupnosti (bod 2) zkopírujeme do paper složky miniatributu, 8. bajt pak do ink složky miniatributu-nezapomeňte ovšem hodnotu barvy před zkopírováním vydělit 32 (z palety 256 barev umíme využít pouze 8, tedy každou 32.).
**6)** Nově vzniklý miniatribut aplikujeme přes výše zkonstruovanou pixelovou část Video RAM.
**7)** Takto pokračujeme až do okamžiku, než zpracujeme všech 49152 bajtů.

Tak, a je to venku. Pokud vám něco není jasné, přečtěte si všechny lekce tohoto seriálu ještě jednou. A pokud pořád nic, počkejte si do příště, uvedeme totiž vzorový příklad této konverze.
Mějte se skvěle, užívejte si léta a těšte se na příště!

**-BLS-**

# Tečka.

## NOVÁ GRAFIKA na Speccy

**...že by přece jen...?**

Od našeno špióna jsme se dozvěděli, že v laboratořích firmy 8BitCompany se nyní pracuje na tajném projektu. Jedná se prý o zařízení nazvané ULA Pro.
Nenechejte se ovšem zmást názvem; není to žádný učitel karate ani odbavovací systém pro rádia. Je to elektronický obvod zapojený do patice čipu ULA přímo ve vašem ZXS.
Instalací ULA Pro získáte:

- -plnohodnotný grafický režim MultiTech (full-screen!)
- -Deprchátor.

V praxi to znamená, že:

- -full-screen MultiColor/MultiTech bude skvěle fungovat-a to bez jakéhokoliv vytěžování procesoru Z80-CPU a nutnosti synchronizace s TV paprskem,
- -již nikdy vám nebudou pršet sovětská dema
- -obě tyto funkce budou vypínatelné a to softwarově!

Náš špionážní fotoaparát zachytil ukázky z připravovaného dema na ULA Pro a to jste ještě neviděli!

P. S.: v laboratořích nám prozradili, že TDI se vyvíjí ve dvou verzích:

- -"Lite" určená pro MB-02+/DataGear,
- -"Advanced" vyráběná právě pro majitele ULA Pro a D40/D80/MB-02+. V této verzi bude použit nejnovější verze MultiTech 7216 rev. 2.23 se skvělými full-screen animacemi!

**-8BC-**

PROTECTION REMOVED by SATANSOFT